Machine Learning Systems: Missing the Forest for the Trees

Mark Zhao, Stanford University

In this new frontier of large-scale machine learning (ML), successful ML deployments now encompass myriad distributed systems and demand expertise across numerous domains – not just architecting, training, and serving a single ML model. For example, industrial training infrastructure requires massive data lakehouses to store datasets, ETL and preprocessing pipelines to ingest data, feature stores to productionize features, and scale-out GPU clusters to train models. Each system is supported by a dedicated team of storage, DB, and/or infrastructure engineers, while ML engineers use these systems to train and fine-tune numerous ML models. While this compartmentalization between systems allows us to manage at-scale deployments, industry teams and academic research largely misses the forest for the trees – being *too* focused on insular optimizations (e.g., how a new ANN algorithm improves vector retrieval throughput by X%) as opposed to understanding their overall impact (e.g., how does improving vector DB retrieval affect overall accuracy and efficiency in a retrieval augmented generation pipeline that combines embedding models/LLMs, vector DBs, prompting techniques, and tool use). To build efficient and impactful ML systems, we need holistic ML systems and frameworks that *a)* allow users (e.g., ML engineers and application developers) to easily program and coordinate complex pipelines of components and *b)* apply cross-cutting optimizations *across components* that reason about and improve end-to-end metrics.

To illustrate opportunities in end-to-end ML systems, we'll use illustrative examples from our experiences at Meta.

a) Need to coordinate compound systems: ML engineers historically managed low-level implementation details across complex systems, resulting in massive inefficiencies. They would directly author data ingestion pipelines (e.g., Spark jobs) to create PB-scale Hive recommendation datasets. To experiment with a new labeling technique, an ML engineer would replicate the pipeline, modify the label column logic (<1% of the dataset size), and land an entirely new table (with >99% duplicate data), resulting in massive storage overheads. Furthermore, when an ML engineer launched a training job, they needed to specify the number of DataLoader workers (with a best-effort guess) to perform the "last-mile" of preprocessing, often resulting in GPU data stalls (if under-provisioned) or underutilized workers (if over-provisioned). ML frameworks should allow ML engineers to author the *logical* pipeline (e.g., feature engineering and labeling logic), while abstracting away the underlying systems details and coordination (e.g., table materialization and worker parallelism) to systems experts.

b) *Need for cross-cutting optimizations:* To continue scaling ML systems, we need to scale *all* ML systems – not just the GPUs. For example, dataset storage and preprocessing infrastructure often require more resources than the GPU training nodes at Meta. To do so, cross-cutting optimizations that coordinate techniques and span across ML pipeline components are essential. For example, we deployed a suite of inter-dependent optimizations spanning ETL, the DataLoader, and the ML model itself to eliminate redundant operations by deduplicating sparse features in recommendation datasets (*RecD* [MLSys'23]). This improved not only training throughput, but also preprocessing and storage efficiency by up to 2.5x, 1.8x, and 3.7x respectively (a massive win at Meta's training scale). To unlock these optimizations, systems builders need to understand the complex interactions and dependencies across the multiple distributed systems that comprise ML pipelines.

Our recent foray into addressing these challenges on the training side is *cedar*. *cedar* is a programming model and framework that allows ML engineers to express logical training input data pipelines by functionally composing native and higher-order operators, abstracting implementation details from users. At the same time, *cedar* easily interfaces with numerous existing execution (e.g., training nodes' CPUs, distributed preprocessing services, etc.) and storage backends (e.g., local/distributed filesystems, cloud data lakes, etc.). This allows *cedar* to coordinate and transparently make cross-cutting optimizations (e.g., operator reordering, materialization, fusion) across its underlying backends using an extensible optimization framework. *cedar* automatically scales input data throughput to support ingestion demand while improving efficiency compared to current state-of-the-art frameworks such as tf.data, Ray Data, and PyTorch DataLoaders by 2.5x, 2.2x, and 2.7x, respectively. Our HPTS talk will describe *cedar*'s ability to manage input data pipelines for large scale training systems, as well as spur discussion and feedback on how we can continue to improve *cedar* (e.g., integration into emerging technologies such as feature stores, or opportunities to enable "data-centric" optimizations such as dataset selection).

At the same time, we also believe our cross-cutting approach has significant merit beyond training data pipelines. For example, LLM applications are increasingly built as pipelines of complex systems including embedding models and LLMs (perhaps multiple), massive vector DBs, prompt optimization techniques, and agents/tool use. There has been significant domain-specific work improving each component (e.g., KV caching systems to improve LLM inference latency/throughput, efficient DBs for vector retrieval, continued NLP research into prompts). However, we want to also leverage HPTS to urge fellow LLM system builders to see the forest and look beyond their domain. How important is a larger LLM if the vector retrieves incorrect information? Does improving LLM serving latency matter if our prompting technique needs many round-trip queries to arrive at the right answer? Fundamentally, how do we (DB experts, system builders, ML engineers) work collectively to continue building large scale ML systems? We hope our talk will instigate a timely debate on this topic.

Short Bio (see also https://web.stanford.edu/~myzhao/)

I am currently a Ph.D. candidate in Electrical Engineering at Stanford University, advised by Christos Kozyrakis.

My research centers around designing systems to improve the scalability, performance, and security of datacenter-scale applications, especially machine learning. I am currently building systems that enable large-scale machine learning training and serving pipelines. My prior research has explored accelerating video analytics pipelines and the security of cloud FPGAs. My research won the IEEE S&P Distinguished Practical Paper Award and is a Top Pick in Hardware and Embedded Security. I am grateful to have been selected as an MLCommons Rising Star. My work is generously supported by a Stanford Graduate Fellowship and a Meta Ph.D. Fellowship in AI System HW/SW Co-Design.

I was also recently a Visiting Researcher in the FAIR and infrastructure teams at Meta from 2020-2022, where I designed, scaled, and optimized data storage and ingestion infrastructure for Meta's machine learning training pipelines. I received my B.S. in Electrical and Computer Engineering from Cornell University in 2018, where I worked with Ed Suh on hardware security.